Does scrum ruin great engineers or are you doing it wrong?

June 29, 2020

A question on Stack Overflow's Software Engineering site caught our attention recently. It tries to come to terms with the impact of scrum on developers' ability to do a great job. The claim is a bold one: Scrum is turning good developers into average ones. Could that be true?

The idea of the scrum framework is to organize a development process to move through the different project cycles faster. But does it always incentivize the right behaviours doing so? Many of the users who joined the debate around the question on Stack Overflow have similar stories of how developers take shortcuts, get

distracted by their ticket high score, or even feign productivity for managers. How can one avoid these pitfalls?

That the question has been migrated from our workplace exchange to the software engineering one shows that developers consider concerns about scrum and its effectiveness larger than the standard software development lifecycle; they feel its effect on their workplace as a whole. User Qiulang makes a bold claim in their question: Scrum is turning good developers into average ones.

Could that be true?

Is scrum the cause of bad development practices or just an excuse for it?

A lot of the debate tried to grapple with the impact and limitations of scrum on any given team or individual. While many see scrum as the cause of team failings, others believe that to be an attribution error and say dysfunction in dev teams runs much deeper.

The scrum defenders see poor management as the cause. Llewellyn writes in his closing remarks: “If management is essentially ignoring the developers, there are fixed deadlines to be achieved with a predefined scope, or it's a dog-eat-dog environment instead of a team focused on

achieving the same goal, if planning ahead and thinking out of the box are not appreciated, then yes, eventually you'll give up and resort to just doing the assigned tasks. I've been there. But don't blame that on scrum."

User DJClayworth echos the sentiment of many comments by saying "developers who think they are under pressure) will do a crappy job in any development methodology."

The opposition is best summarized by user Martin Maat, who points out: "The mere fact that so many people feel the need to say something about it is an indicator of the frustration scrum causes."

Let's unpack some of the points in the discussion. Regardless of whether you are pro or con, we can take a look at the discussion to see where scrum fails developers and where it helps them excel.

What are typical scrum pitfalls?

Whether people are *doing it wrong* or if it is a baked-in bug in the scrum framework, a few common developer traps pop up in the comments. Here are ten that stuck out to us:

1. **Standups are for managers**

A first criticism is about how unintended dynamics happen during standups. One argument is that they can

degenerate to be just as a show of productivity, especially when managers are present. This is the reason user Matthew Gaiser in his answer relabels standups as "update management." In his mind, check-in only invites managers to keep tabs on what is being worked on in unhelpful ways. This is even more true for distributed teams that work asynchronously. (Full disclosure: Matthew has written for us before. But we stumbled across the scrum question by chance)

1. **Prioritising 'done'**

Another point raised is that any process that divides up work and tracks progress leads to a new measurement for progress (deliberate or not). Just by introducing metrics, this influences the behaviour of the people contributing to them.

Many commenters suggest this means developers might cut corners in order to complete what was committed to finish in the current sprint. The problem Gaiser and others point out is that an individual ticket that is being worked on and moved to 'done' during a sprint is a black box. It counts the same for the velocity indicator. "A lousy implementation," Gaiser wirtes "that passes QA and a well-tested, well-architected implementation are equivalent." It's a false indicator of productivity.

1. **Very productive individuals that don't work as a team**

Another thread mulled the discrepancy between great individuals vs. great teams. With everybody following the scrum methodology, you might get some highly efficient developers, but you don't have a team. Gaiser argues that without the right incentives, self-organisation is an unfulfilled goal:

"Team members are going to do things the way they prefer/are incentivized to do and unless that adds up to a useful team, lots of things do not get done and team members just keep marching on over the mess."

Furthermore, leaving each developer to choose their method for solving a problem, said Gaiser, creates more work during debugging.

1. **Complicated tasks get deprioritized**

In a similar vein, Gaiser further criticises the illusion of productivity—there is a focus on getting any ticket to 'done,' while deep thinking does not look very productive. So developers might pick low-hanging fruit and skip the tough-to-solve problems. Gaiser again: "Scrum encourages picking work that can easily be done and rapidly churned out at a steady pace." The result: Daily catch-ups and check-ins encourage picking tasks that can be done in one day.

1. **Features over robust code**

The quality, Gaiser believes, will suffer. "Great developers

are usually defined by their ability to develop robust code. Unless the product owner is technical, scrum massively devalues that as the product owner isn't evaluating the code." To this point he stresses that a 'done ticket' is often a feature decision and not a check of the code being written.

1. **No time for cross pollinating or exchange with peers**

When velocity is the only measurement, the team no longer has time to consult, to give a second opinion, to run a concept by someone—all the things that make a team a team. In Gaiser's words: "Great developers are often sought out for advice and for second opinions. But any time doing that is less time spent churning out tickets, so their velocity falls."

1. **New bugs have to wait**

Another of the scrum bad behaviours is that "bugs are found after the sprint and therefore counted as new work." It incentivises behaviour where developers might release flawed code, because that new work can't be included in the current sprint.

1. **Ticket-driven architecture**

Not only do devs make choices about what to work on based on the ticket system, Gaiser also suggests that scrum unintentionally creates messy architecture, with

developers working through tickets sequentially and independently from each other as “the architecture rapidly begins to mirror the tickets.”

1. **One process to rule them all**

Reading through the discussion you will find those arguing that not following the scrum rules properly enough is the root cause of all the problems. However, perhaps Gaiser’s strongest point against the scrum process is about it becoming the process that overrides everything else, and by this might break a winning team: “[Scrum] bends and breaks every other process to it and becomes this overarching process where you do nothing consistently except scrum rituals and making those scrum rituals seem successful.”

That’s a long list of problems, whether or not they are strictly caused or just brought to the surface by scrum. However, equally numerous were the suggestions of how to allow developers to live up to greatness under the scrum rules.

How to get the most out of scrum?

A lot of the responses to Gaiser’s answer—currently the top-voted—were about how his process was not scrum. Stephen Byrne wrote, "I think this is a good answer with some great ideas, but I must agree with most other commentators, the process being described here is

definitely not scrum as it was meant to be." But enough people either loathe scrum or resonated with Gaiser's answer that something must be broken with how scrum is implemented.

So how can you do scrum right?

Daily standup ≠ new tickets each day

A lot of the debate suggested daily standups create pressure to deliver a closed ticket every day. But prioritization of tasks should still happen As user DJClayworth points out, if this does not happen naturally, it's the job of the scrum master: "You should prioritize the tasks within the sprint, and you should prioritize the big tasks highest, so someone should pick up the big difficult tasks on day one. In any case, if by day two of the sprint, nobody has picked up the big complex task, then the scrum master should say 'I see nobody has started the database compression task—that's a big task and it needs to be started right away if we are going to finish it this sprint.'"

Stop chasing personal bests by not tracking them

Yes, you should break everything in a sprint into small pieces, but scrum does not say you should obsess about the progress—certainly not by pitting devs against each other. Gaiser suggests to stop tracking scores for individuals altogether. He also points out that many managers may need to relearn their process. "Tell

management that the second they praise a dev or give them a raise based on ticket volume, they radically change behavior."

User DJClayworth agrees, actively ignoring individual ticket metrics should be a good scrum master's job: "The focus should be on whether the team completed its commitments as a team. The scrum master should emphasize this and avoid any discussion or measurement of how many stories each person moved."

Break down the big goals, but don't forget about them

Another note on breaking down tasks into chunks: Picking up a small piece of the puzzle does not mean ignoring the puzzle. Llewellyn stresses that the scrum process can be no excuse to forget good software development altogether.

"You should have a good idea of where the project is headed, and you can use this knowledge when you plan the architecture even for the current sprint." Scrum does not absolve people from doing their job as experienced software developers so Llewellyn's plea is to their peers among the readers. "You've been in the planning meetings, you can check the backlog, you know what the overall vision is. You'll want to avoid investing a lot of time into something far ahead in the future, but there's nothing wrong with laying the foundations for an extensible, modular system that works well for what you need right

now and will also support the planned future additions."

Agree on your 'definition of done'

One of the things that pops up is the Definition of Done or DoD and how this helps with keeping standards and expectations for the individual clear. The most pressing questions are who creates it and when? As for the 'when?': *ASAP* or during sprint planning seem to be common suggestions.

For the 'who?' SpoonerNZ writes as reply for another question on Software Engineering. "The Definition of Done is created by the team, but may require the scrum master to enforce quality constraints if the team doesn't have clear development standards. For example, a team may not want code reviews or unit tests, but a scrum master may need to enforce them to ensure quality is maintained. In the ideal situation, the team see the benefits and want such quality constraints, but the real world isn't always ideal."

Who needs to share the DoD? Well, naturally the (difficult) goal is to have everyone on the same team to agree on it. There are good reasons though to extend it to several teams. Or even an organization. Alan Larimer on this: "Without a common definition of 'Done' for a product, quality and its transparency will be negatively impacted. Organizational levels of Definition of Done should be minimal, technical, and sometimes provided by the

organization so it can be applied universally. The organization may provide coding standards. The organization may require automated builds while providing the resources to create and maintain it for each product. Any part of the definition of 'Done' whether created by the organization or by an individual development team must bring value."

Managers as silent observers

Even though it is already part of the scrum guidebook, the discussion seemed to suggest that many, sadly, experience daily standups where managers are present. Because of this, they feel that they need to explain why tickets are taking them longer than they would have to in a meeting with just their peers. So to reiterate: A standup is a team meeting. As stated in the original framework guide: "The daily scrum is an internal meeting for the development team. If others are present, the scrum master ensures that they do not disrupt the meeting."

People over process

If you want a rule about how to apply the set of rules in the scrum framework, Frank Hopkins comments with the classic maxim, "People over processes," and elaborates, "A good team should define its processes, rigid processes don't make a good team."

Another user, meriton, points out that scrum depends on the individuals. "Scrum does not say that developers work

independently. It says that the development team organizes itself, meaning the team gets to decide how its members collaborate."

nvoigt notes that teams in scrum self-organize because they come to a project with a mission already established. "Scrum builds on the fact that you are a team. In a team, it does not matter whether you got 'a ticket done' yesterday. In a team, you agree on what your goal is (i.e. Definition of Done) and then strive to reach it. Together."

Build a team to do scrum, don't expect scrum to build your team.

User nvoigt goes for the sports metaphor: "Just imagine 11 people being handed a soccer manual and being told practice is every day for fifteen minutes around 10 AM in conference room #5. Do you think that is what makes a good soccer team? But what if those 11 people were really good, professional players? Still no team? No. It's just not how you build a team. Just as team sports, scrum needs the participants to be a team. If they are just participants that look to boost their own standing by showing off how many story points they did or how many goals they scored, they will always lose the day to a team that works together instead of next to each other or against each other."

Parting thoughts

nvoigt is ready to admit that scrum does “not work with every person and it does not work in every constellation” and as the participation around the question showed, the set of rules that is scrum might lead to realities that are very far from the intention.

There are many voices in the thread jumping to scrum’s defense and reiterating it as elevating teams to greatness and empowering a team to grow beyond its individuals.

A parting thought comes from Seth R. He says there is no miracle to be expected from agile rituals in general. And demanding them to magically fix a team is asking too much. Rather, his interpretation is: “It's all about tightening up the feedback loop so the team can self-examine and figure out for itself how to get better. Scrum won't help you build a better product, but if you take the self-examination step seriously, it might help you build a better team. That in turn leads to a better product.”

More than one person likened the framework to democracy. So is scrum, perhaps, *the worst form of development process, except for all the others?*

Or is it, as the Scrum guide states, a framework that is simple to understand but difficult to master?